Opt performance

This documents summarizes the main observations about the performance of the implementation of the Regularized Multinomial Regression in `mtool`. This implementation uses a Stochastic Variance Reduced Gradient (SVRG) algorithm in order to solve the proximal operator.

1 Generative Data Model ——

Data are simulated from a $K = 2$ competing risks proportional hazards model. The cause-specific hazard for cause $k \in \{1, 2\}$ for individual i with covariates X_i is $\lambda_k(t|X_i) = \lambda_{0k}(t)\exp(X_i^T \beta_k)$. The baseline hazards $\lambda_{0k}(t)$ follow a Weibull distribution $\lambda_{0k}(t) = h_k \gamma_k t^{\gamma_k - 1}$, with parameters $(h_1, \gamma_1) = (0.55, 1.5)$ and $(h_2, \gamma_2) = (0.05, 1.5)$, implying increasing baseline hazards, higher for cause 1.

The coefficient vectors $\beta_1, \beta_2 \in \mathbb{R}^p$ are sparse. Only the first 10 covariates $X_1, \dots, X_{10}$ have non-zero effects: $\beta_{1j} = +1$ and $\beta_{2j} = -1$ for $j = 1, \dots, 10$, with all other $\beta_{kj} = 0$. These covariates increase $\lambda_1(t|X_i)$ and decrease $\lambda_2(t|X_i)$.

Event times T_i and causes C_i are generated by simulating potential failure times T_{ik} from $\lambda_k(t|X_i)$ and setting $T_i = \min(T_{i1}, T_{i2})$ with C_i being the index yielding the minimum. Independent censoring times $T_{cens,i}$ (rate 0.25) are generated. Observed data consist of $(ftime_i, fstatus_i)$, where $ftime_i = \min(T_i, T_{cens,i})$ and the status $fstatus_i = C_i \cdot \mathbb{1}(T_i \le T_{cens,i})$ (with $fstatus_i = 0$ indicating censoring).

```
###############################################
#' Create the case-base sampled dataset
#'
#' @param surv_obj
#' @param cov_matrix
#' @param ratio
#' @return List of 4 elements, containing the necessary data to fit a case-base
#'   regression model.
create_cbDataset <- function(surv_obj, cov_matrix, ratio = 5) {
 n <- nrow(surv_obj)
 B <- sum(surv_obj[, "time"])
 c1 <- sum(surv_obj[, "status"] == 1)
 c2 <- sum(surv_obj[, "status"] == 2)
 b <- ratio * (c1)
 offset <- log(B/b)
 prob_select <-  surv_obj[, "time"]/B
 # Create base series
```

```r
  which_pm <- sample(n, b, replace = TRUE, prob = prob_select)
  bSeries <- as.matrix(surv_obj[which_pm, ])
  time_bseries <- runif(b) * bSeries[, "time"]
  cov_bseries <- cov_matrix[which_pm, , drop = FALSE]
  event_bseries <- rep(0L, nrow(bSeries))
  # Extract case series
  cSeries <- as.matrix(surv_obj[surv_obj[,"status"] != 0L, ])
  time_cseries <- cSeries[,"time"]
  cov_cseries <- cov_matrix[surv_obj[,"status"] != 0L, , drop = FALSE]
  event_cseries <- cSeries[,"status"]
  # Combine and return
  output <- list("time" = c(time_bseries, time_cseries),
                 "event_ind" = c(event_bseries, event_cseries),
                 "covariates" = rbind(cov_bseries, cov_cseries),
                 "offset" = rep(offset, nrow(bSeries) + nrow(cSeries)))

  return(output)
}

########### Function to compute weibull hazard ################################
weibull_hazard <- Vectorize(function(gamma, lambda, t) {
  return(gamma * lambda * t^(gamma - 1))
})

############ Function to simulate from cause-specific hazards #################
cause_hazards_sim <- function(p, n, beta1, beta2,
                              nblocks = 4, cor_vals = c(0.7, 0.4, 0.6, 0.5),
                              num.true = 20, h1 = 0.55, h2 = 0.10,
                              gamma1 = 100, gamma2 = 100, max_time = 1.5,
                              noise_cor = 0.1,
                              rate_cens = 0.05, min_time = 0.002,
                              exchangeable = FALSE) {
  # Warnings
  if(length(beta1) != length(beta2)) stop("Dimension of beta1 and beta2 should be the same")
  if(nblocks != length(cor_vals)) stop("Dim of nblocks and corr for blocks should match")
  if(isTRUE(exchangeable)) {
    # Create an empty matrix
    mat <- matrix(noise_cor, nrow = p, ncol = p)
    # Set the correlation values
    cor_exchangeable <- 0.5
    # Set the upper triangular and lower triangular parts
    mat[1:num.true, 1:num.true] <- cor_exchangeable
    # Print the matrix
    diag(mat) <- rep(1, length(diag(mat)))
    X <- mvtnorm::rmvnorm(n, mean = rep(0, p), sigma = mat)
  } else {
    # Set the number of variables per block
    vpb <- num.true/nblocks
```

```r
# Set the correlation values for each covariate block
correlation_values <- cor_vals
# Initialize empty matrix
correlation_matrix <- matrix(noise_cor, nrow = p, ncol = p)
# Generate the covariance matrix with block correlations
for (i in 1:nblocks) {
  start_index <- (i - 1) * vpb + 1
  end_index <- i * vpb
  correlation_matrix[start_index:end_index, start_index:end_index] <- correlation_values[
}
# Diagonal elements should be 1
diag(correlation_matrix) <- rep(1, length(diag(correlation_matrix)))
X <- mvtnorm::rmvnorm(n, mean = rep(0, p), sigma = correlation_matrix)
}
X <- as.matrix(X)
# Specify rate parameters
lambda1 <- h1 * exp(X %*% beta1)
lambda2 <- h2 * exp(X %*% beta2)
# Define cdf - U
cdf_U <- function(t, gamma1, lambda1, gamma2, lambda2, U) {
  F_min_U <- 1 - exp(-(lambda1 * t^gamma1 + lambda2 * t^gamma2)) - U
  return(F_min_U)
}
# Generate uniform values and store in dataframe
u <- stats::runif(n)
# Inverse transform sampling
dat_roots <- cbind.data.frame(u, gamma1, lambda1, gamma2, lambda2)
times <- dat_roots %>%
  dplyr::rowwise() %>%
  dplyr::mutate(
    t_tilde = stats::uniroot(
      cdf_U,
      interval = c(.Machine$double.eps,
                   max_time),
      extendInt = "yes",
      U = u,
      gamma1 = gamma1,
      lambda1 = lambda1,
      gamma2 = gamma2,
      lambda2 = lambda2
    )$`root`
  ) %>%
  dplyr::pull(t_tilde)
# Generate event indicators
hazard1 <- weibull_hazard(gamma= gamma1, lambda = lambda1, t = times)
hazard2 <- weibull_hazard(gamma = gamma2, lambda = lambda2, t = times)
event <- stats::rbinom(n = n, size = 1, prob = hazard1 / (hazard1 + hazard2))
c.ind <- ifelse(event == 1, 1, 2)
```

```r
  # Add censoring
  cens <- stats::rexp(n = n, rate = rate_cens)
  c.ind <- ifelse(cens < times, 0, c.ind)
  times <- pmin(cens, times)
  # Winsorize time ranges to desired ones to make them more realistic
  # and add some white noise
  c.ind <- ifelse(times >= max_time, 0, c.ind)
  times <- ifelse(times >= max_time, max_time, times)
  times[times == max_time] <- times[times == max_time] +
    rnorm(length(times[times == max_time]), mean = 0, sd = 1e-4)
  times <- ifelse(times < min_time, min_time, times)
  times[times == min_time] <- times[times == min_time] +
    abs(rnorm(length(times[times == min_time]), mean = 0, sd = 1e-4))
  sim.data <- data.frame(fstatus = c.ind, ftime = times)
  X <- as.data.frame(X)
  colnames(X) <- paste0("X", seq_len(p))
  sim.data <- as.data.frame(cbind(sim.data, X))
  return(sim.data)
}

  gen_data <- function(n, p) {
      num_true <- 20
      beta1 <- c(rep(0, p))
      beta2 <- c(rep(0, p))
      nu_ind <- seq(num_true)
      # Here out of 20 predictors, 10 should be non-zero
      beta1[nu_ind] <- c(rep(1, 10), rep(0, 10))
      beta2[nu_ind] <- c(rep(-1, 10), rep(0, 10))

      # Simulate data
      sim.data <- cause_hazards_sim(n = n, p = p,
                              beta1 = beta1, beta2 = beta2,
                              rate_cens = 0.25,
                              h1 = 0.55, h2 = 0.05,
                              gamma1 = 1.5, gamma2 = 1.5,
                              exchangeable = TRUE)


      cen.prop <- c(prop.table(table(sim.data$fstatus)), 0, 0, 0, 0)

      # Training-test split
      # We only do this (instead of generating datasets for train and test
      # like Anthony mentioned because it is faster computationally
      # as casebase resamples) + proportion of censoring can be quite random
      # in each run of the simulation so we want to maintain the same in
      # validation and test set

      train.index <- caret::createDataPartition(sim.data$fstatus, p = 0.75, list = FALSE)
```

```
  train <- sim.data[train.index,]
  test <- sim.data[-train.index,]

  ############################################################
  # We have two competitor models for variable selection:
  # 1) Independent cox-regression model
  # 2) penCR cox regression model - where the lambda penalties are trained together
  ######################## Fit indepedent cox-regression model ########################
  ######################### Cause-1 ##########################################
  # Censor competing event
  y_train <- Surv(time = train$ftime, event = train$fstatus == 1)

  x_train <- model.matrix(~ . -ftime -fstatus, data = train)[, -1]

  # Censor competing event
  y_test <- Surv(time = test$ftime, event = test$fstatus == 1)

  x_test <- model.matrix(~ . -ftime -fstatus, data = test)[, -1]

  # Test set
  surv_obj_val <- with(test, Surv(ftime, as.numeric(fstatus), type = "mstate"))

  # Covariance matrix
  cov_val <- cbind(test[, c(grepl("X", colnames(test)))], time = log(test$ftime))

  # Case-base dataset
  cb_data_val <- create_cbDataset(surv_obj_val, as.matrix(cov_val), ratio = 10)


  return(list(X_train = cb_data_val$covariates,
         Y_train = cb_data_val$event_ind,
         offset = cb_data_val$offset,
         true_beta = cbind(beta1, beta2)))
}
```

2 Performance Evaluation ——

In order to evaluate the performance of `MNlogistic`, we assess its convergence time and memory allocated with sample sizes of $n = 1000, 2000$ and covariates $p = 20, 100, 250, 500, 1000, 2000, 4000$. The original implementation is compared with an improved implementation `MNlogistic2` with the following changes:

- Matrix input `U` in `proximal` functions (Flat, Graph, Tree) are directly called using `U.memptr()`
- String `regul` in `proximal` functions (Flat, Graph, Tree) are directly called using `std::vector<char>`.
- Multinomial gradient functions changed to in-place calculation (takes `arma::mat& grad_out`, returns `void`), removing internal allocation.

- Multinomial gradient functions optimized to create column vector from x only once per call.

```
if (save){
  results <- bench::press(
    n = c(1000, 2000),
    p = c(20, 100, 250, 500, 1000, 2000, 4000),
    lambda = c(0.9, 0.5, 0.07, 0.005),
    {
      data <- gen_data(n, p)
      # lambda <- 0.9
      alpha = 0.5
      unpen_cov = 2
      # Elastic-net reparametrization
      lambda1 <- lambda*alpha
      lambda2 <- 0.5*lambda*(1 - alpha)
      # Prepare covariate matrix with intercept
      set.seed(1234)
      bench::mark(
        MNlogistic = mtool::mtool.MNlogistic(
          X = data$X_train,
          Y = data$Y_train,
          offset = data$offset,
          N_covariates = 2,
          regularization = 'elastic-net',
          transpose = FALSE,
          lambda1 = lambda1, lambda2 = lambda2,
          lambda3 = 0
        ),
          MNlogistic2 = mtool::mtool.MNlogistic2(
          X = data$X_train,
          Y = data$Y_train,
          offset = data$offset,
          N_covariates = 2,
          regularization = 'elastic-net',
          transpose = FALSE,
          lambda1 = lambda1, lambda2 = lambda2,
          lambda3 = 0
        ),
        glmnet = glmnet(data$X_train,
                        data$Y_train,
                        alpha = alpha,
                        lambda = lambda,
                        family = "multinomial"),
        min_iterations = 1, check = FALSE
      )
    }
  )
```

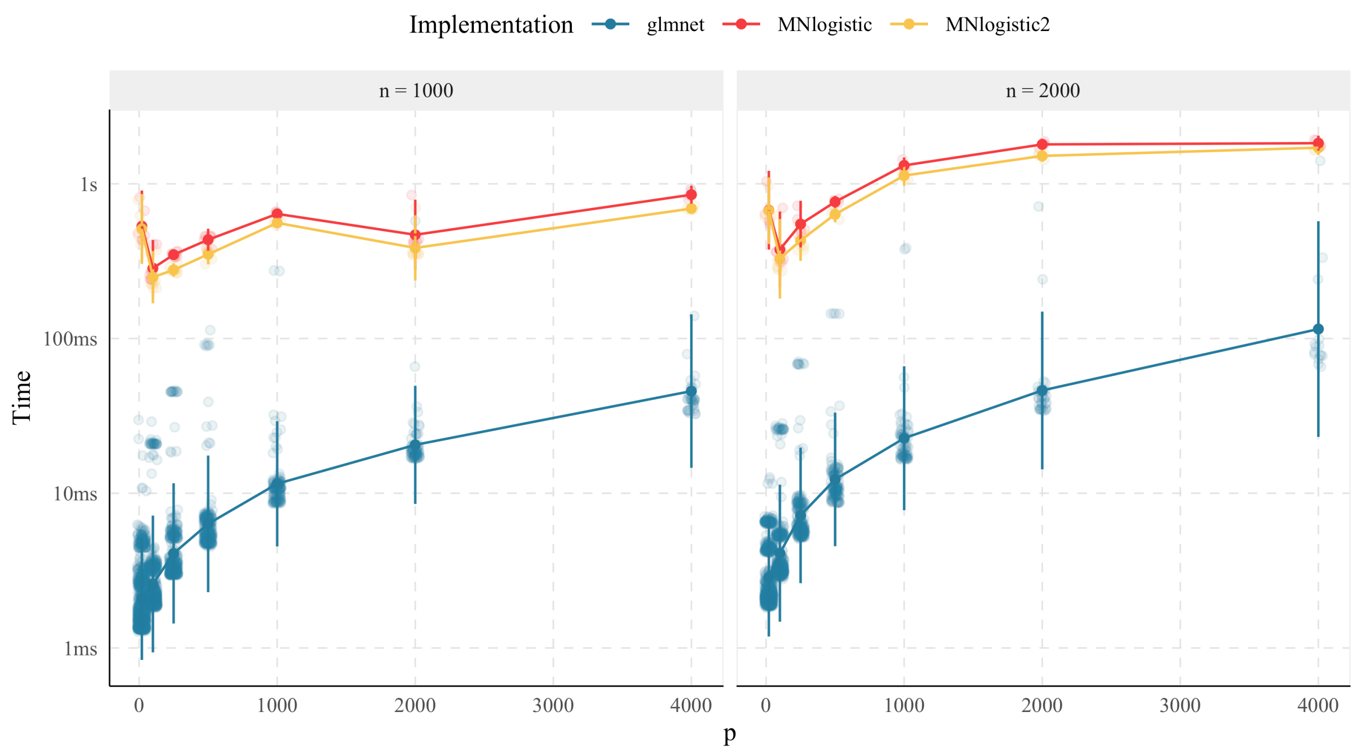
```
  pushoverr::pushover(message = "Loop finished",
                      user = "uk2t3aqv1tavuxiuuzag7pf7742q15",
                      app = "aq5nckww93an2onvb4k1jin9487gak")


  saveRDS(results, here::here( "notes_jmr", "data", "results.rds"))
}
```

2.1 Plot results

2.1.1 Time

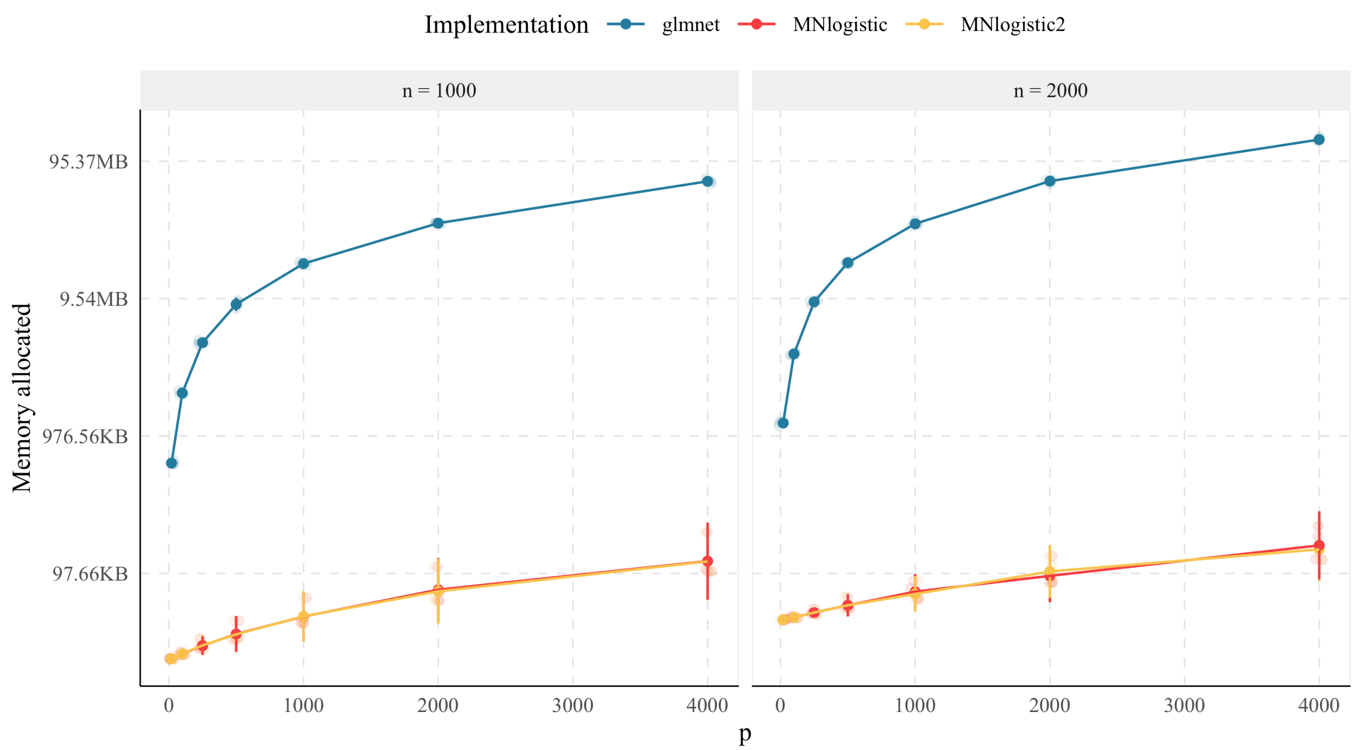
```
results <- readRDS(here::here("notes_jmr", "data", "results.rds"))

results %>%
  as.tibble() %>%
  unnest_longer(time) %>%
  ggplot(aes(x = p, y = time, group = paste0(expression, p, n),
             colour = as.character(expression))) +
  geom_jitter(alpha = 0.1) +
  stat_summary(geom = "point", fun.y = "mean", alpha = 1) +
  stat_summary(fun.data = mean_sdl, geom = "errorbar", width = 0.1) +
  stat_summary(aes(group = paste0(expression, n)), fun = mean, geom = "line") +
  facet_wrap(~paste0("n = ", n)) +
  labs(x = "p",
       y = "Time",
       colour = "Implementation")
```

```
#> Warning: `as.tibble()` was deprecated in tibble 2.0.0.
#> i Please use `as_tibble()` instead.
#> i The signature and semantics have changed, see `?as_tibble`.

#> Warning: The `fun.y` argument of `stat_summary()` is deprecated as of ggplot2 3.3.0.
#> i Please use the `fun` argument instead.
```



2.1.2 Memory

```
results %>%
  as.tibble() %>%
  ggplot(aes(x = p, y = mem_alloc, group = paste0(expression, p, n),
         colour = as.character(expression))) +
  geom_jitter(alpha = 0.1) +
  stat_summary(geom = "point", fun.y = "mean", alpha = 1) +
  stat_summary(fun.data = mean_sdl, geom = "errorbar", width = 0.1) +
  stat_summary(aes(group = paste0(expression, n)), fun = mean, geom = "line") +
  facet_wrap(~paste0("n = ", n)) +
  labs(x = "p",
      y = "Memory allocated",
     colour = "Implementation")
```



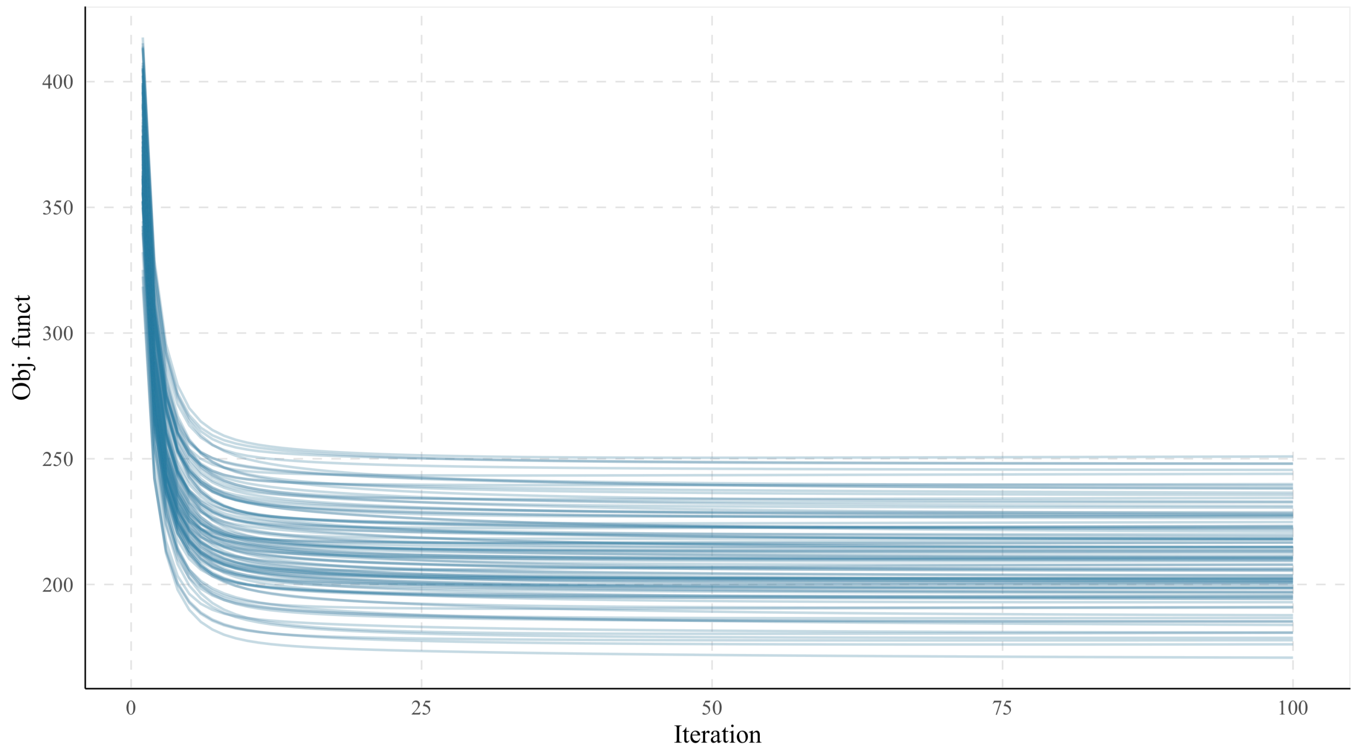
2.2 Optimization Example

```
# Objetive function

objective <- function(B, X, Y, offset,
                lambda, alpha, reg_p) {


   N_prime <- nrow(X)
   p <- ncol(X)
   K <- ncol(B)
   Y_int <- as.integer(Y)
   Y <- Y_int
   valid_y_indices <- which(Y >= 1 & Y <= K)
   if (length(valid_y_indices) < N_prime) {
      if (length(valid_y_indices) == 0) {
         loss_value <- 0.0

         penalty_value <- 0.0
         if (reg_p > 0 && lambda > 0) {
            B_reg <- B[1:reg_p, , drop = FALSE]
            lambda1 <- lambda * alpha
            lambda2 <- 0.5 * lambda * (1 - alpha)
            penalty_value <- (lambda1 * sum(abs(B_reg))) + (lambda2 * sum(B_reg^2))
         }
         return(penalty_value)
      }
```

```r
    X_filt <- X[valid_y_indices, , drop = FALSE]
    Y_filt <- Y[valid_y_indices]
    offset_filt <- offset[valid_y_indices]
    N_filt <- length(valid_y_indices)
  } else {
    # Use all data if all Y are valid
    X_filt <- X
    Y_filt <- Y
    offset_filt <- offset
    N_filt <- N_prime
  }

  # eta = X %*% B + offset (N_filt x K)
  eta <- sweep(X_filt %*% B, 1, offset_filt, "+")

  # log(1 + sum_k(exp(eta_ik))) for each row i
  max_eta_stable <- apply(cbind(0, eta), 1, max) # max of 0 and row max(eta_k)
  log_one_plus_sum_exp <- max_eta_stable + log( exp(-max_eta_stable) + rowSums(exp(sweep(et

  idx_mat <- cbind(seq_len(N_filt), Y_filt)
  eta_yi <- eta[idx_mat]

  #  -eta_iyi + log(1 + sum_k exp(eta_ik))
  loss_per_obs <- -eta_yi + log_one_plus_sum_exp

  loss_value <- sum(loss_per_obs)

  # Penalty
  penalty_value <- 0.0
  if (reg_p > 0 && lambda > 0) {

    B_reg <- B[1:reg_p, , drop = FALSE]

    lambda1 <- lambda * alpha
    lambda2 <- 0.5 * lambda * (1 - alpha)

    l1_norm <- sum(abs(B_reg))

    l2_norm_sq <- sum(B_reg^2)

    penalty_value <- (lambda1 * l1_norm) + (lambda2 * l2_norm_sq)
  }

  return(loss_value + penalty_value)
}
```

```r
if (save){
set.seed(123)

iter_sim <- tibble()

for (i in 1) {
n <- 1000
p <- 20
lambda = c(1e-10)
data <- gen_data(n, p)
alpha = 0.5
unpen_cov = 2
# Elastic-net reparametrization
lambda1 <- lambda*alpha
lambda2 <- 0.5*lambda*(1 - alpha)

MNlogistic3 <- mtool::mtool.MNlogistic3(
   X = data$X_train,
   Y = data$Y_train,
   offset = data$offset,
   N_covariates = unpen_cov,
   regularization = 'elastic-net',
   transpose = FALSE,
   lambda1 = lambda1, lambda2 = lambda2,
   lambda3 = 0,
   niter_inner_mtplyr = 2, maxit = 100,
   tolerance = 1e-10
)

MNlogistic3$coefficients

iter_vals <- map_dbl(MNlogistic3$coefficientshist, ~
                objective(matrix(.x,
                             nrow = ncol(data$X_train)),
                        data$X_train, data$Y_train, data$offset,
                        lambda, alpha, 20))

iter_sim <- tibble(obj_step = iter_vals,
     iter = 1:length(iter_vals),
     sim = i) %>%
bind_rows(iter_sim)
}

saveRDS(iter_sim, here::here( "notes_jmr", "data", "opt-example.rds"))

}

iter_sim <- readRDS(here::here("notes_jmr", "data", "opt-example.rds"))
```

```
iter_sim %>%
  ggplot(aes(iter, y = obj_step, group = sim)) +
  geom_line(alpha = 0.3) +
  # geom_smooth(aes(group = 1)) +
  labs(y = "Obj. funct",
     x = "Iteration")
```



2.2.1 Comparison

```
if (save){

for (i in 1:30) {

  seed <- as.integer(i)

  # take the last five digits of the initial seed
  the_seed = seed %% 100000

  set.seed(the_seed)
  n <- 1000
  p <- 2000
  lambda = c(1e-10)
  data <- gen_data(n, p)
  alpha = 0.5
  unpen_cov = 2
  # Elastic-net reparametrization
  lambda1 <- lambda*alpha
```

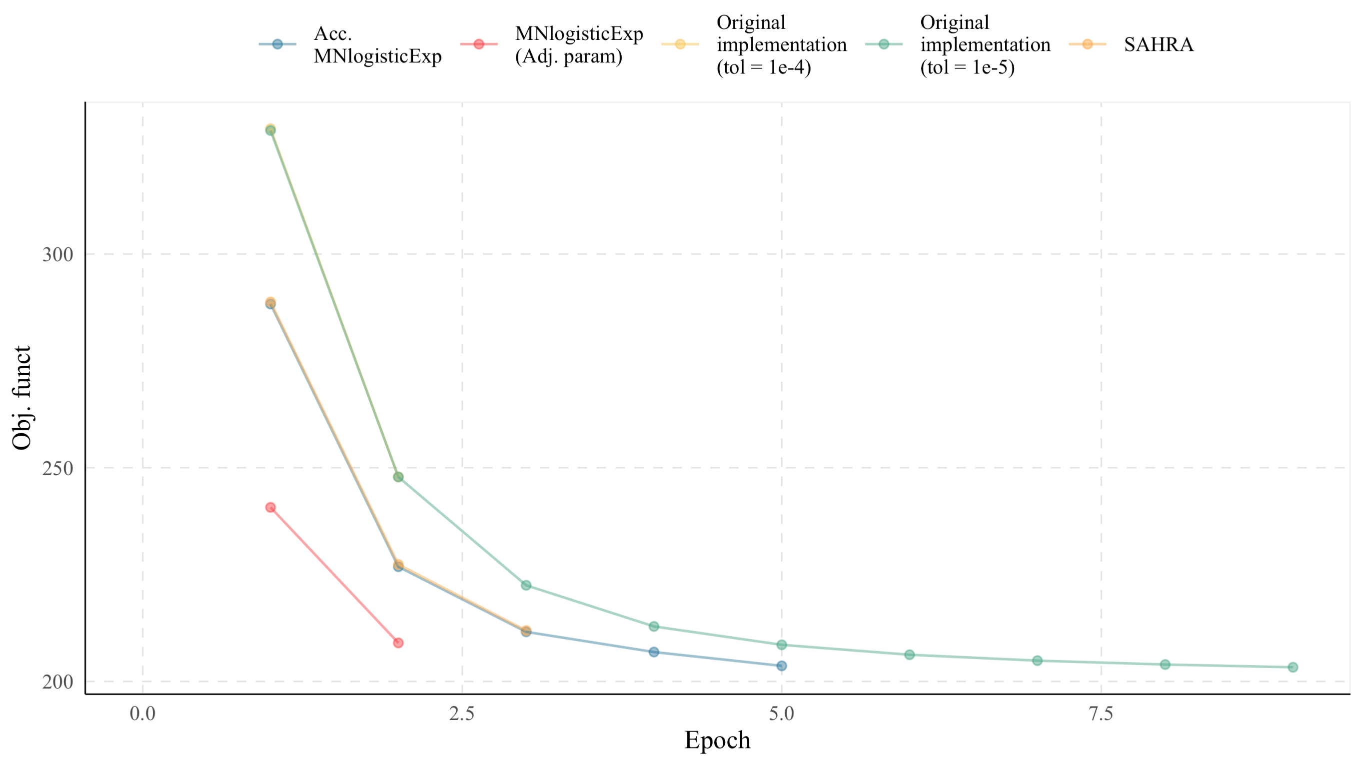
```r
lambda2 <- 0.5*lambda*(1 - alpha)


set.seed(the_seed)
time <- system.time({MNlogistic <- mtool::mtool.MNlogistic3(
  X = data$X_train,
  Y = data$Y_train,
  offset = data$offset,
  N_covariates = unpen_cov,
  regularization = 'elastic-net',
  transpose = FALSE,
  lambda1 = lambda1, lambda2 = lambda2,
  lambda3 = 0,
  niter_inner_mtplyr = 7, maxit = 100,
  tolerance = 1e-4
)})

time_tol <- system.time({MNlogistic_tol <- mtool::mtool.MNlogistic3(
  X = data$X_train,
  Y = data$Y_train,
  offset = data$offset,
  N_covariates = unpen_cov,
  regularization = 'elastic-net',
  transpose = FALSE,
  lambda1 = lambda1, lambda2 = lambda2,
  lambda3 = 0,
  niter_inner_mtplyr = 7, maxit = 100,
  tolerance = 1e-5
)})


set.seed(the_seed)
timeExp <- system.time({MNlogisticExp <- mtool::mtool.MNlogisticExp(
  X = data$X_train,
  Y = data$Y_train,
  offset = data$offset,
  N_covariates = unpen_cov,
  regularization = 'elastic-net',
  transpose = FALSE,
  lambda1 = lambda1, lambda2 = lambda2,
  lambda3 = 0,
  niter_inner_mtplyr = 2, maxit = 100,
  tolerance = 1e-4,
  learning_rate = 1e-3
)})

set.seed(the_seed)
timeAcc <- system.time({MNlogisticAcc <- mtool::mtool.MNlogisticAcc(
```

```r
    X = data$X_train,
    Y = data$Y_train,
    offset = data$offset,
    N_covariates = unpen_cov,
    regularization = 'elastic-net',
    transpose = FALSE,
    lambda1 = lambda1, lambda2 = lambda2,
    lambda3 = 0,
    niter_inner_mtplyr = 1,
    maxit = 50,
    momentum_gamma = .9,
    tolerance = 1e-1,
    learning_rate = 1e-4,
    pos = T
  )})

  set.seed(the_seed)
  timeSAHRA <- system.time({MNlogisticSAHRA <- mtool::mtool.MNlogisticSAHRA(
    X = data$X_train,
    Y = data$Y_train,
    offset = data$offset,
    N_covariates = unpen_cov,
    regularization = 'elastic-net',
    transpose = F,
    lambda1 = lambda1, lambda2 = lambda2,
    lambda3 = 0,
    niter_inner_mtplyr = 1,
    maxit = 50,
    tolerance = 1e-4,
    learning_rate = 1e-3
  )})


  iter_vals <- map_dbl(MNlogistic$coefficientshist, ~
                  objective(matrix(.x,
                                   nrow = ncol(data$X_train)),
                            data$X_train, data$Y_train, data$offset,
                            lambda, alpha, 20))

  iter_vals_tol <- map_dbl(MNlogistic_tol$coefficientshist, ~
                      objective(matrix(.x,
                                       nrow = ncol(data$X_train)),
                                data$X_train, data$Y_train, data$offset,
                                lambda, alpha, 20))
  iter_valsExp <- map_dbl(MNlogisticExp$coefficientshist, ~
                     objective(matrix(.x,
                                      nrow = ncol(data$X_train)),
                               data$X_train, data$Y_train, data$offset,
```

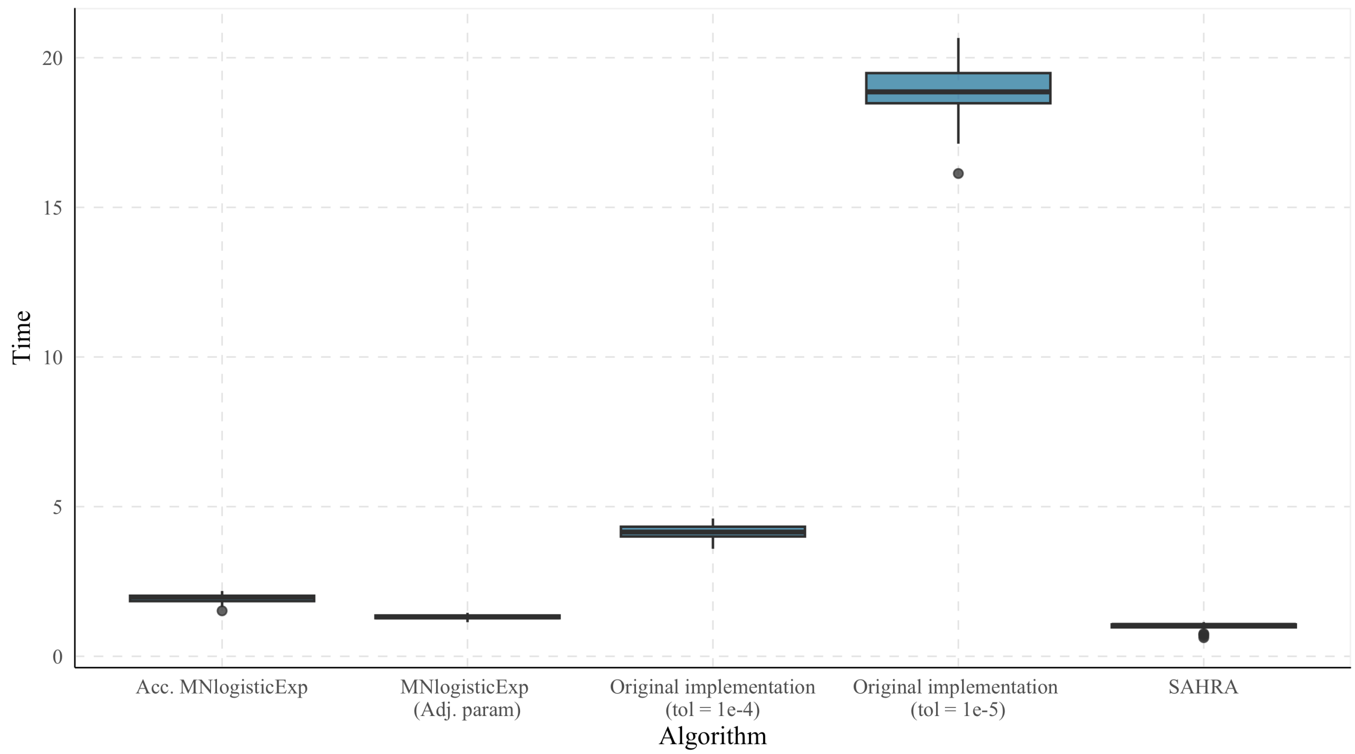
```
                              lambda, alpha, 20))

  iter_valsAcc <- map_dbl(MNlogisticAcc$coefficientshist, ~
                      objective(matrix(.x,
                                    nrow = ncol(data$X_train)),
                              data$X_train, data$Y_train, data$offset,
                              lambda, alpha, 20))

   iter_valsSAHRA <- map_dbl(MNlogisticSAHRA$coefficientshist, ~
                      objective(matrix(.x,
                                    nrow = ncol(data$X_train)),
                              data$X_train, data$Y_train, data$offset,
                              lambda, alpha, 20))

  # MNlogistic$coefficients
  # MNlogisticExp$coefficients
    if (i == 1) iter_sim <- tibble()

  iter_sim <- tibble(obj_step = iter_vals,
                  iter = 1:length(iter_vals),
                  sim = i,
                  time = time[1],
                  algorithm = "Original implementation\n(tol = 1e-4)") %>%
    bind_rows(tibble(obj_step = iter_vals_tol,
                   iter = 1:length(iter_vals_tol),
                   sim = i,
                   time = time_tol[1],
                   algorithm = "Original implementation \n(tol = 1e-5)"),
          tibble(obj_step = iter_valsExp,
                   iter = 1:length(iter_valsExp),
                   sim = i,
                   time = timeExp[1],
                   algorithm = "MNlogisticExp \n(Adj. param)"),
          tibble(obj_step = iter_valsAcc,
                   iter = 1:length(iter_valsAcc),
                   sim = i,
                   time = timeAcc[1],
                   algorithm = "Acc. MNlogisticExp"),
          tibble(obj_step = iter_valsSAHRA,
                   iter = 1:length(iter_valsSAHRA),
                   sim = i,
                   time = timeSAHRA[1],
                   algorithm = "SAHRA")) %>%
    bind_rows(iter_sim)
  saveRDS(iter_sim, here::here( "notes_jmr", "data", "opt-algorithms.rds"))
}
```

```r
}

 iter_sim <- readRDS(here::here("notes_jmr", "data", "opt-algorithms.rds"))

  iter_sim %>%
    ggplot(aes(iter, y = obj_step, group = paste0(algorithm, sim),
               color = str_wrap(algorithm, 15))) +
    # geom_line(alpha = 0.2) +
    stat_summary(aes(group = algorithm), geom = "point", fun.y = "mean", alpha = 0.5) +
    stat_summary(aes(group = algorithm), fun = mean, geom = "line", alpha = 0.5) +
    labs(y = "Obj. funct",
     x = "Epoch",
     color = "") +
    xlim(0,9)
```

```
#> Warning: Removed 4 rows containing non-finite outside the scale range
#> (`stat_summary()`).
#> Removed 4 rows containing non-finite outside the scale range
#> (`stat_summary()`).
```




```r
  iter_sim %>%
    group_by(sim, algorithm) %>%
    summarise(time = first(time)) %>%
  ggplot(aes(x = algorithm, y = time)) +
    geom_boxplot() +
    labs(y = "Time",
       x = "Algorithm")
```



Notes:

- Inner loop iterations do not appear to increase the number of epochs proportionally, suggesting that it may reduce computational time. The inner loop iterations set to 2n appear to perform effectively, reducing computational time to 1/3.
  - As the value of p increases, it seems that the algorithm fails to reach the minimum.
  - The situation becomes problematic when $p >> n$ because the initial epoch is below the tolerance level and far from the optimal position. To address this issue, we can lower the tolerance; however, this increases the number of steps required. By reducing the tolerance, we can observe that it becomes necessary to achieve reasonable coefficients in this context.
- **Tolerance** of 1e-4 in high-dimensional settings is not enough to reach optimal levels.
  - Maybe, we can reduce the number of iterations as we increase the number of epochs.
  - Under similar tolerance, an smaller tolerance seems to be better.
- **Learning rate**. The increase in learning rate from 1e-4 to 1e-3 reduces computational time by half. We should consider better learning rates. A smaller learning rate generates NaNs in the Frobenius norm.